VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY

DEPARTMENT OF RADIO ELECTRONICS

APLIKACE PRO ROZPOZNÁVÁNÍ LOGA AUTOMOBILOVÉ ZNAČKY

APPLICATION FOR CAR LOGO RECOGNITION

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Tomáš Uchytil

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Petr Kadlec, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Elektronika a komunikační technologie**

Ústav radioelektroniky

Student: Tomáš Uchytil ***ID:*** 230337

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Aplikace pro rozpoznávání loga automobilové značky

POKYNY PRO VYPRACOVÁNÍ:

Proveďte rešerši dostupných neuronových sítí a vyberte síť vhodnou pro rozpoznávání automobilového loga na základě fotografie automobilu [1]. Vytvořte databázi fotografií kapot automobilů obsahující loga automobilů. V databázi budou zastoupeny fotografie s různou kvalitou, natočením, světelnými podmínkami a alespoň 10 různými automobilovými značkami.

Vybranou neuronovou síť natrénujte pomocí vytvořené databáze fotografií [2]. Neuronovou síť otestujte na datech, která nebyla použita pro trénování. Vytvořte aplikaci pro mobilní telefony (ideálně pro Android i iOS), která umožní vybrat dostupný obrázek (vyfotografovat automobil) a použije natrénovanou neuronovou síť pro určení značky automobilu.

DOPORUČENÁ LITERATURA:

[1] WEIDMAN, Seth. Deep learning from scratch: building with Python from first principles. Beijing: O'Reilly, [2019]. ISBN 9781492041412.

[2] MICHELUCCI, Umberto. Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection. 1. USA: Apress, 2019. ISBN 978-1484249758.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 29.5.2023

Vedoucí práce: doc. Ing. Petr Kadlec, Ph.D.

doc. Ing. Lucie Hudcová, Ph.D.
předseda rady studijního programu

ABSTRACT

The thesis deals with finding a suitable neural network for recognizing a brand of a car by its logo and programming and training this network. This is then implemented into a mobile application that recognizes the logo on a newly taken photo or on an image selected from the mobile phone's storage.

KEYWORDS

Logo recognition, Convolutional neural network, Car logo, Python, Kivy, Mobile app, Android

ABSTRAKT

Práce se zabývá nalezením vhodné neuronové sítě pro rozpoznání loga automobilové značky a implementací a trénováním této sítě. Ta je následně implementována do mobilní aplikace, která umožňuje rozpoznání loga na základě nově pořízené fotografie, nebo na základě obrázku vybraného z úložiště mobilního telefonu.

KLÍČOVÁ SLOVA

Rozpoznání loga, konvoluční neuronová síť, automobilové logo, Python, Kivy, mobilní aplikace, Android

ROZŠÍŘENÝ ABSTRAKT

Neuronové sítě jsou počítačové modely, které simulují procesy lidského mozku, například rozpoznávání předmětů podle jejich identifikačních znaků, rozhodování o dalším úkolu počítače nebo tvorbu uměleckých děl.

Cílem této práce je fotografování log různých automobilových značek a jejich systematické označení. Dalším krokem je zjišťování informací o neuronových sítích a výběr sítě nejvhodnější pro dané použití. Posledním krokem této práce je trénování vybrané sítě pomocí vybraných fotografií.

V práci jsem nejdříve rozebral pojmy jako rozpoznání vzorce neuronovou sítí, konvoluční neuronové sítě, datasety pro neuronové sítě nebo matice záměn (confusion matrix). Vysvětlení jsou doplněna ilustračními obrázky pro jednodušší pochopení tématu.

Prvním praktickým krokem bylo získání fotografií pro naučení neuronové sítě. Podařilo se mi získat přibližně 500 nově vzniklých fotografií, 1800 fotografií získaných z datasetu CompCars a 2900 fotografií z datasetu Kaggle dostupných online.

Následovalo hledání vhodné neuronové sítě. Po důkladné rešerši dostupných neuronových sítí jsem se rozhodl pro model popsaný v práci [1]. Ačkoli tento model pracuje s videem, jádro neuronové sítě je vhodné pro rozpoznávání statických obrazů automobilových log. Proto ve své práci využívám část kódu z [1] upravenou pro práci s jpg obrázky.

Kód jsem také doplnil o program pro roztřídění datasetu do složek podle značky automobilu (tento formát je potřebný pro knihovnu k naučení neuronové sítě), a o program pro generování matice záměn pro posouzení přesnosti neuronové sítě. Dále jsem si napsal program pro hrubé ořezání fotografií, aby obsahovaly převážně logo - kvůli zvýšení přesnosti detekce a snížení potřebného výpočetního výkonu. Dále jsem vytvořil konvoluční neuronovou síť dosahující úspěšnosti přibližně 80 %.

V další fázi jsem vytvořil aplikaci pro operační systém Android, kterou jsem umístil do obchodu s aplikacemi Google Play. Aplikace umožňuje vybrat existující fotografii z galerie, nebo pořídit novou fotografii, a následně zobrazí detekovanou značku.

Pro tvorbu aplikace jsem si vybral framework Kivy, který umožňuje převod kódu v programovacím jazyce python, na jazyk Java nebo Swift pro mobilní platformy. Pro tento účel existuje více frameworků, ale Kivy se vyznačuje širokou komunitou zajišťující dobrou technickou podporu, a také je jedním z nejjednodušších na programování.

V průběhu tvorby aplikace jsem využil několik již existujících knihoven pro python: Kivy pro tvorbu samotné aplikace, KivyMD pro grafické prvky, Pillow pro načítání a úpravu obrázků, numpy pro nalezení nejvyšší hodnoty v poli pravděpodobností vráceném neuronovou sítí, os pro získání cesty souborů a ověření existence

souborů, camera4kivy pro komunikaci s fotoaparátem, knihovny base64 a requests pro volitelné odesílání špatně rozpoznaných fotografií na můj server pro budoucí vylepšení neuronové sítě, dále knihovna Plyer pro uživatelsky přívětivější volbu souborů a PyJNIus pro komunikaci s operačním systémem.

Pro kompilaci aplikace pro operační systém Android se v případě frameworku Kivy využívá nástroj Buildozer v příkazovém řádku operačního systému Linux. Tento nástroj umožňuje nejen export aplikací pro přímou instalaci do mobilního telefonu, ale také tvorbu balíčků (Android app bundle - aab) pro publikování v obchodě s aplikacemi.

Kompilace pro iOS probíhá na platformě OSX na počítačích společnosti Apple. Nejprve je v příkazovém řádku pomocí nástroje Toolchain z python kódu vytvořen projekt, který je následně možné otevřít a kompilovat v aplikaci jménem XCode pro tentýž operační systém. Pro publikování v obchodě aplikací pro iOS je nutné pravidelné roční předplatné vývojářského účtu, ale aplikaci je možné do lokálního zařízení instalovat bezplatně pomocí XCode studia.

Při tvorbě aplikace jsem musel vytvořit uživatelské rozhraní. Typický životní cyklus aplikace je následující: po spuštění aplikace se zobrazí načítací obrazovka, během níž probíhají na pozadí přípravné úkony. Následně je tato obrazovka nahrazena hlavním menu. Zde má uživatel možnost volby, zda vybere fotografii z galerie, nebo pořídí novou. V prvním případě se zobrazí nativní nástroj pro výběr souborů, ve druhém obrazovka fotoaparátu. Po vybrání nebo pořízení fotografie proběhne rozpoznání, a uživatel je přenesen na obrazovku s výsledkem. Pokud zobrazené logo nesouhlasí s realitou, má uživatel možnost kliknout na tlačítko "incorrect logo", čímž odešle nesprávně rozpoznanou fotografii na vzdálený server a na 3 sekundy se zobrazí děkovná hláška. Použitím tlačítka zpět v levém horním rohu, případně systémového tlačítka zpět, se uživatel z jakékoliv obrazovky vrátí zpět do hlavního menu.

Aplikace dále nabízí možnost personalizace pomocí obrazovky nastavení, kam se uživatel dostane stisknutím ikony s ozubeným kolečkem v levém horním rohu hlavní obrazovky. Tato nastavení obsahují volbu tématu (světlé nebo tmavé) a zapnutí vývojářského režimu, který zobrazí některé dodatečné informace, jako cestu a rozlišení obrázku, na hlavní obrazovce. Tyto informace mohou následně posloužit pro hledání chyb, pokud se na některých zařízeních aplikace nechová dle očekávání. Při normální funkci aplikace se vývojářský režim nepoužije.

V době tvorby mobilní aplikace nebylo možné vytvořit aplikaci pro iOS, protože nástroj Toolchain, který je zodpovědný za převod Kivy kódu na Xcode projekt, ve verzi Apple clang 14.0.3 není kompatibilní s nástrojem Xcode, který kompiluje aplikaci do jazyka swift pro iOS, ve verzi 14.3 (14E222B). Až bude tento problém vyřešen ze strany vývojářů zmíněných nástrojů, je uživatelské rozhraní a většina kódu díky použití multiplatformního frameworku Kivy připravena pro tvorbu ap-

likace pro iPhone, bude pouze nutné doplnit některé funkce specifické pro tento operační systém, například oprávnění nebo detekci připojení k internetu.

Natrénovanou konvoluční neuronovou síť nebylo nakonec v mobilní aplikaci možné využít z důvodu omezení vyplývajících z architektury a výpočetního výkonu mobilních zařízení. Byla tedy vytvořena nová, jednodušší síť, která ale není tolik přesná. Nejlepších výsledků tedy aplikace dosahuje pro značky Opel, Škoda a Toyota, kde byl k dispozici dostatek kvalitních fotografií. V případě ostatních značek by pro vyšší přesnost bylo nutné rozšířit dataset tréninkových fotografií.

UCHYTIL, Tomáš. *Application for car logo recognition*. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Radio Electronics, 2023, 49 p. Bachelor's Thesis. Advised by doc. Ing. Petr Kadlec, Ph.D.

Author's Declaration

Author:	Tomáš Uchytil
Author's ID:	230337
Paper type:	Bachelor's Thesis
Academic year:	2022/23
Topic:	Application for car logo recognition

I declare that I have written this thesis independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author, I furthermore declare that, with respect to the creation of this thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to thank my supervisor, doc. Ing. Petr Kadlec, Ph.D. for his support and advice, and for pointing me in the right direction during the making of this thesis.

Contents

List of Figures

Introduction

Neural networks are computer models, that simulate procedures of the human brain, e.g. classify objects depending on their identification marks, decide the next action of a computer or create an artificial artwork.

The goal of this thesis is to take pictures of logos for multiple car brands. Find out more about neural networks, and then find one that is the most suitable to classify car logos. Train chosen neural network using the photos taken, as well as some photos found online.

For easier use of the neural network, a new mobile app will be developed, implementing the trained neural network and allowing users to take a new picture, or select an existing one from the gallery, which will then automatically identify a logo on said picture.

Kivy multiplatform framework will be used for app development. App will be divided into two parts, app-related data, which will be installed together with the app, and AI model, which will be downloaded separately once the app is launched. This way, the app and the model can be updated separately, making updates easier and limiting the amount of data needed to be downloaded.

1 Neural Networks

A neural network consists of several elements connected together. It tries to simulate the behavior of the human brain and biological nervous systems by extend. Its function is to produce an output pattern based on input parameters. In this thesis, pattern classification is used, specifically [2]. Before we get into details, it's necessary to explain some neural network terms.

- Neuron - Elementary structure of a neural network. It processes input information and generates outputs based on its trained parameters.
- Weights - It says, how much influence each neuron has on the outputs of the network.
- Layer - set of neurons with same the parameters and function. There are three main types of layers in a neural network. There is a visible input layer receiving input and passing it over, visible output layer taking care of generating outputs and hidden middle layers performing specific tasks and passing processed data to the next layers.
- Neural network - net of interconnected layers with associated weights passing data between themself, being tuned based on the network's experience.
- Training - A process of finding the best combination of neuron and layer parameters by presenting images to the neural network, and then rewarding it for correct classification and penalizing for mistakes [3].

1.1 Pattern Classification

When trying to classify a set of images into certain categories based on what they represent, you can't just simply check, how many bits and bytes they have in common or try overlapping them and checking for similarities, because objects you're trying to recognize can have a quite wide variation. What we have to do to achieve this is to separate input data into groups, each representing one of the objects we want to classify. When we talk about neural networks, we call these groups classes. In this case, our classified object doesn't need to match the others exactly, but it is instead decided which class the image pattern is most similar to [2].

A typical use for neural network pattern classification is a handwritten letter/number recognition (used for example for automatic letter sorting by Czech Post), object recognition (can be used by security cameras for detecting intruders), face recognition (usable for example when searching for criminals or missing people), and many more. In this thesis, pattern classification will be used to recognize different car logos given to the neural network as input, and select car brands, which will be determined by the neural network as an output. The typical structure of the

neural network is in Fig. 1.1. For our purpose, there will be one input containing a picture of the logo, multiple hidden layers, and 23 outputs, one for each car brand.

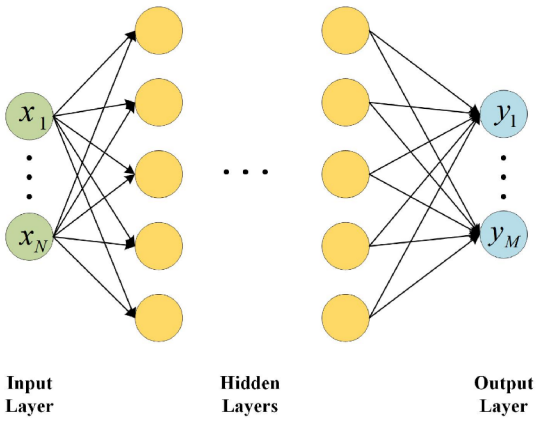

Fig. 1.1: Neural network structure [4].

1.2 Convolutional Neural Networks

A key benefit of Convolutional neural network (CNN) over Artificial neural network (ANN) is the reduction of the number of parameters, that the neural network needs. Also, CNN is able to detect patterns regardless of size and orientation, saving one step of image preprocessing. In the first layers, it detects the most distinct features of the image, like sharp edges and corners. Going on, simple shapes are detected, and then even deeper, some finer, high-level features are obtained [5]. An example of what CNN can see is shown in Fig. 1.2.

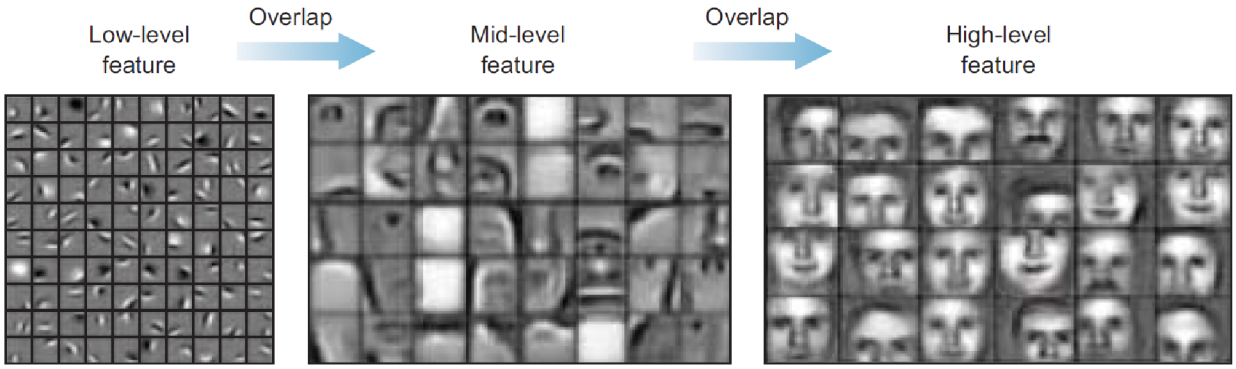

Fig. 1.2: Learned features from a convolutional neural network [6].

1.3 Datasets for Neural Networks

A dataset is a file or a folder containing data used to train and validate a neural network. A very simple and small dataset for the recognition of hand-written digits is shown in Fig. 1.3. There are several key characteristics of high-quality datasets. They are:

- QUALITY - To avoid false classification, you need to make sure you supply your neural network with a high-quality dataset.
- QUANTITY - To properly train your neural network model, you need to supply it with as much data as possible.
- USABILITY - Good dataset should be easy to work with. It should be divided into folders, properly labeled and all data should have the same format.
- SCALABILITY - You should be able to add data to your dataset on the go. There is a possibility you may need to add another class to your dataset in the future or risk overfitting your model (training on a small dataset causing your model to work well on recognizing data from a dataset but badly on real world data). In this case, you need to extend your dataset and train your network again.

Depending on what type of neural network you are creating, you can use many dataset formats. You can see some of them in the list below [7]:

- TEXT/NUMERIC
  - Probably the easiest one to work with. Since you don't usually work with extremely long numbers or sentences, you can quite easily store them in a comma-separated text file (CSV). Then, you have both data and labels in one file making it very convenient for neural network learning.
  - Typical use cases are AI chatbots, data analysis, etc.
- AUDIO
  - Slightly trickier, since it can't be stored in one file. However, you still work with a one-dimensional data stream.
  - Typical use cases are speech recognition, music recognition, noise removal, etc.
- IMAGE
  - When talking about images, we usually talk about two-dimensional data. In the easiest case, the image is black and white, therefore data stream only consists of luminance information. If you want to work with the color images, you have to process two more "layers" of the given pictures.
  - Typical use cases are optical character recognition (OCR), image editing

(denoising, object removal...), image generating, or, as used in this thesis, it can be used to recognize objects in the image.

- VIDEO
  - Probably the hardest format to process. We usually don't treat video as one object representing movement, but instead as a series of individual images.
  - Typical use cases are motion tracking, 2D or 3D rendering, video processing, etc.

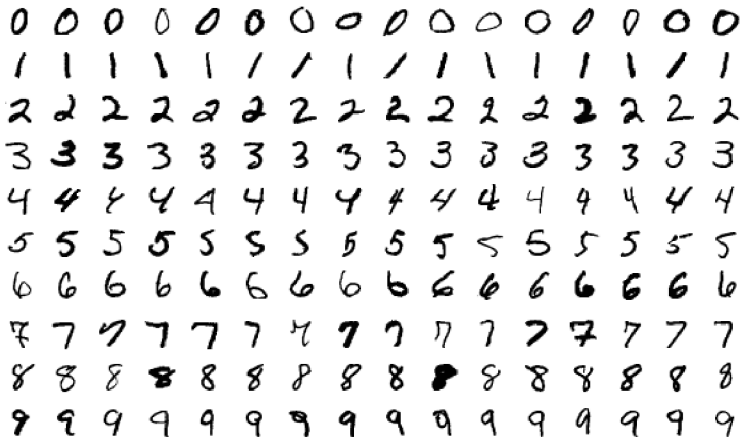

Fig. 1.3: Example of a dataset for handwritten digits (MNIST dataset) [8].

1.4 Confusion Matrix

There are many ways to assess the accuracy of neural networks. The easiest one is the percentage of correctly recognized data. Errors, however, are usually not evenly distributed across all classes. Instead, some class pairs are more prone to be mixed up. Therefore, it is useful to generate not only the average accuracy but also a matrix that shows which classes have been confused with which during the test. This is called a confusion matrix. Such a matrix provides much more detailed information on the results of the test than the mere error rate. It shows which classes were classified properly or almost properly and which were misclassified/confused with other classes and to what degree [9].

In example Fig. 1.4, you can see the confusion matrix of the neural network for emotion classification. The given network is more accurately detecting angry and happy emotions, compared to sad and neutral. Another important data is the most common misclassification is sad emotion being recognized as neutral.

		Predicted			
		Angry	Happy	Sad	Neutral
Actual	Angry	**95.88%**	1.19%	0.48%	2.46%
	Happy	1.76%	**94.49%**	0.91%	2.85%
	Sad	1.22%	3.50%	**83.37%**	11.91%
	Neutral	5.62%	6.71%	6.39%	**81.28%**

Fig. 1.4: Example of the confusion matrix (emotion detection) [10].

2 Practical Implementation

For implementation, I used the programming language Python, which is widely used for machine learning and neural networks. Firstly, I gathered some photos of car logos. Then I found and modified a neural network to classify logos. I chose a convolutional neural network, because the car logos I need to recognize contain lots of sharp edges and corners, therefore using CNN should reduce neural network parameters compared to ANN, and remove dependency on logo position and angle, as discussed in [5]. Hopefully, due to fewer parameters of CNN, this will show the best results for the computational power given. This neural network was then trained and its success rate was verified using a confusion matrix.

2.1 Photo Database

I got photos from the internet, my phone camera, and some photos of less common brands were taken by my family members. A list of photos sorted by source can be seen in Tab. 2.1.

2.1.1 Own Photos

I managed to get about 495 photos of 23 car brands on my own [11]. Photos I took cover different light scenarios, as well as different logo angles, so these factors hopefully won't influence the accuracy of the final app. Also, some brands changed their logo in the past, so I tried to include as many variants as possible. The influence of angle and light can be seen in Fig. 2.1. When taking pictures, I didn't find a lot of Teslas, so hopefully, there will be enough samples to accurately represent this brand.

Fig. 2.1: Example of different quality photos of my own.

2.1.2 Datasets

I found two datasets containing desired photos of car logos. First of them is the Comprehensive Cars (CompCars) [12] dataset. It contains 136,726 photos. However, most of the photos don't include a logo, and the ones that do, contain a lot of car brands, which do not occur in the Czech Republic. After deleting photos not suitable for neural network training, I am left with about 1800 photos of 23 car brands.

Another dataset is the Kaggle Car Brand Logos dataset. Kaggle is a website unifying various tools for machine learning. There are online courses, Q&A blog, and place, where users can share and download datasets, to name a few. This data set contains photos of eight car brands. It's divided into train and test images. In the training set, there are over 300 images per brand. In the test set, there are 63 photos for each brand. There are 2562 photos in total [13].

Tab. 2.1: Number of photos for each car brand

Car brand	Own photos	compCars	Kaggle	Total
Audi	18	148	0	166
BMW	21	158	0	179
Citroen	20	64	0	84
Fiat	16	26	0	42
Ford	26	66	0	92
Hyundai	19	83	352	454
Chevrolet	7	90	0	97
Kia	24	86	0	110
Mazda	14	70	367	451
Mercedes	23	136	392	551
Mitsubishi	6	62	0	68
Nissan	7	73	0	80
Opel	21	15	351	387
Peugeot	26	81	0	107
Renault	18	29	0	47
Seat	48	3	0	51
Skoda	78	61	364	503
Subaru	10	45	0	55
Suzuki	10	72	0	82
Tesla	6	1	0	7
Toyota	23	108	356	487
Volkswagen	38	213	380	631
Volvo	9	103	0	112
Total	488	1793	2562	4843

2.2 Neural Network

After a thorough search of available neural networks, we decided to use the model described in the bachelor's thesis of Bc. Marek Sicha [1]. After examination of his code, I've come to the conclusion that his code, while performing a different task, can be modified to fit my application.

2.2.1 Code Modifications

To reuse found code, I had to make the following changes:

- Original code worked with .ppm image format. While images could be converted to this format quite easily, my project is supposed to work with images imported from the user's mobile phone or taken by the phone's camera, and these are usually in .jpg format. Therefore, I changed the input image format to .jpg to eliminate unnecessary conversion.
- I moved the image size, which was hardcoded as 32x32 pixels in [1], to the global variable. This allows me to experiment with different image sizes/resolutions to find the most reliable combination for the best classification results.
- I replaced the model definition with one found in [14]. I found better results using this model with approximately 90% accuracy over about 60% using model based on [1] (results after only training the compCars dataset).
- I had to write some Python functions to quickly sort dataset photos into folders and crop them to only contain logos, which is the only part of the photo I'm concerned about, with regard to this project.
- I also created a function to generate a confusion matrix to evaluate neural network accuracy. This function takes the trained model and image path as input. It then classifies each image in a path with a trained neural network and compares the result to the known brand name. Finally, it outputs the number of correctly and incorrectly recognized photos. Once the neural network was modified and trained (using only the compCars dataset for easier changes and faster feedback), I generated a confusion matrix, which can be seen in Fig. 2.2.

	Audi	Bmw	Citroen	Fiat	Ford	Hyundai	Chevrolet	Kia	Mazda	Mercedes	Mitsubishi	Nissan	Opel	Peugeot	Renault	Seat	Skoda	Subaru	Suzuki	Tesla	Toyota	Volkswagen	Volvo
Audi	92,59%	3,70%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,70%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Bmw	0,00%	98,73%	0,00%	0,00%	0,00%	0,00%	0,63%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,63%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Citroen	0,00%	0,00%	87,50%	0,00%	1,56%	3,13%	1,56%	0,00%	0,00%	1,56%	0,00%	0,00%	0,00%	3,13%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	1,56%	0,00%	0,00%
Fiat	0,00%	0,00%	0,00%	96,15%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	3,85%	0,00%
Ford	0,00%	0,00%	0,00%	0,00%	81,82%	3,03%	4,55%	1,52%	0,00%	1,52%	1,52%	0,00%	0,00%	6,06%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Hyundai	0,00%	0,00%	0,00%	0,00%	1,20%	91,57%	2,41%	1,20%	0,00%	0,00%	0,00%	2,41%	0,00%	1,20%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Chevrolet	0,00%	1,11%	0,00%	0,00%	2,22%	0,00%	96,67%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Kia	0,00%	1,16%	0,00%	1,16%	1,16%	1,16%	4,65%	83,72%	0,00%	0,00%	1,16%	1,16%	0,00%	3,49%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	1,16%	0,00%
Mazda	0,00%	0,00%	0,00%	0,00%	1,43%	1,43%	1,43%	1,43%	82,86%	0,00%	0,00%	5,71%	0,00%	2,86%	0,00%	0,00%	0,00%	0,00%	2,86%	0,00%	0,00%	0,00%	0,00%
Mercedes	0,00%	0,00%	0,74%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	94,12%	1,47%	1,47%	0,00%	0,00%	0,74%	0,00%	0,00%	0,00%	1,47%	0,00%	0,00%	0,00%	0,00%
Mitsubish	0,00%	1,61%	0,00%	1,61%	0,00%	0,00%	1,61%	0,00%	0,00%	1,61%	88,71%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	4,84%	0,00%	0,00%	0,00%	0,00%
Nissan	0,00%	0,00%	0,00%	0,00%	1,37%	0,00%	0,00%	0,00%	1,37%	0,00%	0,00%	89,04%	0,00%	6,85%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	1,37%
Opel	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	6,67%	0,00%	0,00%	13,33%	66,67%	13,33%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Peugeot	0,00%	1,23%	0,00%	0,00%	1,23%	1,23%	0,00%	1,23%	0,00%	0,00%	0,00%	0,00%	0,00%	95,06%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Renault	0,00%	3,45%	0,00%	3,45%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	79,31%	0,00%	0,00%	0,00%	6,90%	0,00%	3,45%	3,45%	0,00%
Seat	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Skoda	0,00%	4,92%	0,00%	0,00%	1,64%	0,00%	0,00%	3,28%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	86,89%	0,00%	1,64%	0,00%	1,64%	0,00%	0,00%
Subaru	0,00%	0,00%	0,00%	0,00%	0,00%	8,89%	2,22%	0,00%	0,00%	0,00%	0,00%	4,44%	4,44%	6,67%	0,00%	0,00%	0,00%	71,11%	0,00%	0,00%	2,22%	0,00%	0,00%
Suzuki	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	1,39%	0,00%	0,00%	0,00%	2,78%	0,00%	0,00%	0,00%	0,00%	95,83%	0,00%	0,00%	0,00%	0,00%
Tesla	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	100,00%	0,00%	0,00%	0,00%
Toyota	0,00%	0,93%	0,00%	0,00%	0,93%	1,85%	0,00%	1,85%	0,00%	0,00%	0,00%	1,85%	0,00%	2,78%	0,93%	0,00%	0,00%	0,00%	1,85%	0,00%	86,11%	0,93%	0,00%
Volkswage	0,00%	0,00%	0,00%	0,47%	0,47%	0,00%	0,00%	0,47%	0,00%	0,00%	0,00%	0,00%	0,00%	2,35%	0,00%	0,00%	0,00%	0,00%	0,47%	0,00%	0,00%	95,77%	0,00%
Volvo	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,97%	0,00%	0,97%	0,97%	0,97%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	96,12%

Fig. 2.2: Confusion matrix of the neural network pretratined using compCars dataset only.

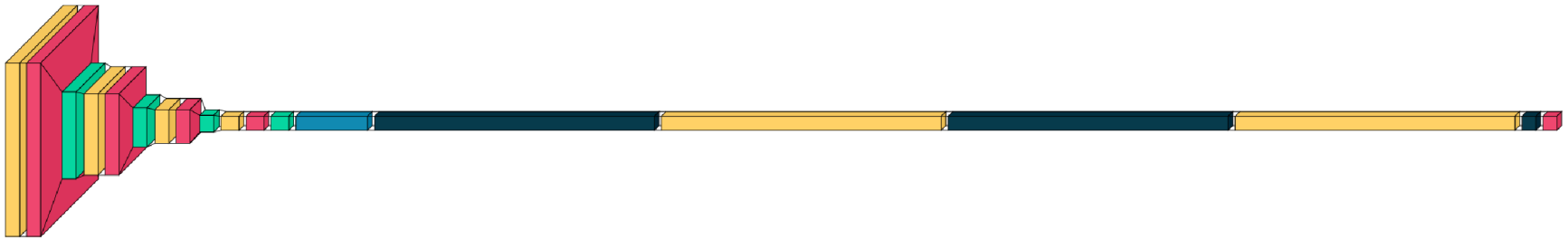

Fig. 2.3: Layered view of the neural network I'm using.

2.3 App development

The goal of this thesis was to create a multiplatform mobile app to classify a picture of a car logo, and since I opted for using python for my neural network, I had to find a framework allowing me to convert python code to code required by all mobile platforms in question. That's Java or Kotlin for Android and Object-C for iOS.

2.3.1 Framework

For such a framework, I had a few choices, each with its advantages and disadvantages.

- **Kivy** is probably the easiest framework to work with python code-wise. It includes a python interpreter in the compiled app, which allows it to work with most of the available python libraries. On the other hand, it brings a larger app size because these resources are not very optimized for the target platform. This framework requires Linux to compile for Windows and Android, and OS-X for iOS compilation. It can be debugged by installing required libraries and then running source code on Windows with Python installed.
  Kivy has been available since 2011, therefore it's a well verified framework [15].
- **BeeWare** is more developer friendly, compared to Kivy, and it uses platform-specific UI toolkit, so it looks and feels like a native app. The main disadvantage of BeeWare is, that it's still quite a new framework. Therefore, it doesn't have as good documentation as other frameworks, and very few python libraries are supported [15].
- **Django** is very strong for backend programming, however, it provides very little frontend-wise. It requires another framework like React native or Kivy (mentioned above). It's mainly used in applications dealing mainly with online data transfers [16].

For some reasons mentioned above, I decided to use the KIVY framework. Mainly because I really needed to use python libraries (TensorFlow library for neural network, at the very least) and I didn't want to combine different frameworks for frontend and backend, which could result in some compatibility problems in development.

I decided to overcome one of the biggest disadvantages of Kivy (namely its terrible non-native and vintage-like UI) by including a UI library called KivyMD. MD in the name stands for Material design, which is a design system by Google. This library quite successfully ports this design to the Kivy framework, and in addition, allows one-line theme changes. This library also adds a bit of the native-like feel

mentioned above.

2.3.2 Libraries used for app development

Various tasks I was trying to achieve were already solved by other people before, therefore, there are so-called libraries available for use. In programming, a library is a module responsible for performing specific activities. Here are some of the libraries I used with a brief description of their purpose.

- **Kivy** - Framework responsible for making an app out of Python code. This library was discussed in more detail in the chapter above.
- **KivyMD** - Material design module for Kivy. It makes the app and its user interface look more modern, i.e. creating rounded shaded buttons, instead of plain rectangles. It also adds some new controls, like a top bar with an optional action button on the left side.
- **Pillow** - Image library used to load images, and also for resizing them to AI-required size and making them grayscale.
- **numpy** - The AI model returns a result as an array of probabilities in the order in which they were trained. Therefore, the numpy library is used to find a class with the highest probability and return the ID of the selected class. It's also used to convert image output from Pillow to a float-type array for a neural network.
- **os** - This library is used to get the path of the main Python file, and also to check for the existence of a given file.
- **camera4kivy** - Library responsible for communication with the device camera. Unlike other similar libraries, this one is a very universal platform-vise and also allows pinch-to-zoom gestures on the preview.
- **base64 and requests** - Two libraries with a common purpose. When the user marks the logo recognition as incorrect (more on that later), base64 turns an image into a base64 encoded string, which is then, with the help of the requests library, sent to a server using an encrypted HTTPS POST request.
- **Plyer** - It is used to call the device default file-chooser. It would be possible to develop an in-app file selector without another library, but it would require more permissions to be requested, and I think it would feel more natural and private for users to use the same file selector across the whole OS.
- **PyJNIus and android** - Libraries directly communicating with OS on the lower Java layer. In my app, it is used to get the Android version from the system and ask for permissions (In Android 13 and higher, the app must use run-time permission requests instead of just declaring it needs to use them).

2.3.3 Tensorflow lite

For neural network to work on mobile devices, Tensorflow has to be converted to a simpler Tflite model. Because this conversion limits the number of model operators, conversion can provide various results. Since I wasn't happy with the quality of the converted Tensorflow AI model, I settled on creating a new Tflite one, based on the pre-trained Keras Sequential_1 model. With this approach, accuracy increased from 65% to 80%. Detailed results can be seen in Fig. 2.4. As you can see, the new neural network has fairly fewer incorrect classifications, and most of the incorrect ones come from Fiat, Tesla, and Subaru, which all had quite a small dataset. Part of the result could also be a difference between input shapes of neural networks, with most of the images in the test dataset being about 300x300 pixels, which could mean that the old network might outperform the new one when dealing with lower quality photos, but simply not being as good at it with higher quality ones.

Old tensorflow model (input size 128 x 128)

Input class (correct) \ Output class (result of classification)	Audi	Bmw	Citroen	Fiat	Ford	Hyundai	Chevrolet	Kia	Mazda	Mercedes	Mitsubishi	Nissan	Opel	Peugeot	Renault	Seat	Skoda	Subaru	Suzuki	Tesla	Toyota	Volkswagen	Volvo
Audi	75.00%	0.00%	0.00%	0.00%	5.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	10.00%	0.00%
Bmw	0.00%	40.00%	0.00%	5.00%	0.00%	10.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	40.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Citroen	0.00%	0.00%	85.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Fiat	0.00%	5.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	10.00%	60.00%	0.00%
Ford	5.00%	10.00%	0.00%	0.00%	60.00%	10.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Hyundai	0.00%	0.00%	0.00%	0.00%	0.00%	95.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Chevrolet	0.00%	0.00%	0.00%	0.00%	5.00%	10.00%	50.00%	0.00%	0.00%	0.00%	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	5.00%	0.00%	5.00%	0.00%	5.00%	0.00%	10.00%
Kia	15.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	55.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	5.00%	5.00%	0.00%
Mazda	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	95.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Mercedes	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	5.00%	85.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Mitsubishi	0.00%	0.00%	10.00%	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	30.00%	35.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	10.00%	0.00%
Nissan	0.00%	0.00%	0.00%	20.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	5.00%	0.00%	50.00%
Opel	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	85.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%
Peugeot	0.00%	5.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	10.00%	65.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%
Renault	0.00%	0.00%	0.00%	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	55.00%	0.00%	5.00%	0.00%	0.00%	0.00%	5.00%	20.00%	0.00%
Seat	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	10.00%	5.00%	0.00%	65.00%	10.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Skoda	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Subaru	0.00%	0.00%	0.00%	0.00%	30.00%	5.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	10.00%	40.00%	0.00%	0.00%	5.00%	0.00%	0.00%
Suzuki	0.00%	5.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	5.00%	5.00%	60.00%	0.00%	0.00%	0.00%	0.00%
Tesla	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	5.00%	0.00%	5.00%	25.00%	30.00%	20.00%	0.00%
Toyota	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	90.00%	0.00%	0.00%
Volkswagen	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%
Volvo	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	95.00%

New tflite model (input size 224 x 224)

Input class (correct) \ Output class (result of classification)	Audi	Bmw	Citroen	Fiat	Ford	Hyundai	Chevrolet	Kia	Mazda	Mercedes	Mitsubishi	Nissan	Opel	Peugeot	Renault	Seat	Skoda	Subaru	Suzuki	Tesla	Toyota	Volkswagen	Volvo
Audi	95.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Bmw	0.00%	90.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%
Citroen	0.00%	0.00%	95.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%
Fiat	0.00%	0.00%	0.00%	35.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	55.00%	0.00%
Ford	0.00%	0.00%	0.00%	0.00%	85.00%	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Hyundai	0.00%	0.00%	0.00%	0.00%	0.00%	85.00%	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%
Chevrolet	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	90.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Kia	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	85.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%
Mazda	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Mercedes	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	95.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Mitsubishi	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	10.00%	0.00%	0.00%	10.00%	55.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	20.00%	0.00%
Nissan	0.00%	0.00%	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	55.00%	10.00%	5.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%
Opel	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	95.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Peugeot	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Renault	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	0.00%	65.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%	0.00%
Seat	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	75.00%	0.00%	0.00%	10.00%	0.00%	0.00%	5.00%	0.00%
Skoda	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	95.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%
Subaru	0.00%	0.00%	0.00%	5.00%	0.00%	5.00%	0.00%	15.00%	0.00%	10.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	5.00%	50.00%	0.00%	0.00%	0.00%	0.00%	5.00%
Suzuki	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	90.00%	0.00%	0.00%	0.00%	0.00%
Tesla	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	10.00%	10.00%	0.00%	0.00%	0.00%	10.00%	0.00%	0.00%	5.00%	0.00%	5.00%	40.00%	0.00%	20.00%	0.00%
Toyota	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	95.00%	5.00%	0.00%
Volkswagen	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	95.00%	0.00%
Volvo	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%	0.00%	0.00%	0.00%	80.00%

Fig. 2.4: Comparison of the old and new AI models.

2.4 App creation

Since every mobile platform has its specifics, it is necessary to compile universal Python code to Java code for Android OS, and Swift code for iOS.

2.4.1 Android compilation

For the Kivy framework, the tool for compilation is called Buildozer, and it needs to run on Linux OS. The compilation process is quite simple. You just need to fill out some parameters of the app, like name, version, required permissions, architectures to compile for, etc. Buildozer itself is controlled via a command line, with the possibility to export a debug app with an apk extension for direct installation, or a signed app bundle with an aab extension to publish your app in the Google Play store.

2.4.2 iOS compilation

The theoretical compilation process for iOS apps goes as follows: you need macOS device (either Apple MAC or Macbook), where you install Python with the Toolchain library. Toolchain should be able to convert Kivy source files to XCode project (XCode is Apple's proprietary app builder tool required to make apps for iOS). After that, you open the project in XCode and compile the app. However, when I tried to follow this process myself, I run into some issues when opening generated project in Xcode. I think this has to do with some compatibility problems between version 14.3 (14E222B) of Xcode and version Apple clang 14.0.3 of Toolchain. Unfortunately, I can't use an older version of XCode, because then I wouldn't be able to compile apps for the new version of iOS, and iPhones can't be easily downgraded to test my app. Therefore, at the time of writing this thesis, I decided it would be best to abandon iOS development attempts for now and focus on debugging the Android version of the app.

3 App user interface

Since I created the app, I had to design a user interface for it as well. Navigation through the app is illustrated in Figure 3.1, where black and yellow arrows are showing usual application flow, red arrows demonstrate arrows pointing back to home screen and green arrow shows, where users are taken when they press the cogwheel icon and go to settings.

3.1 Loading screen

This is the initial screen that users see when they launch the app. When this screen is displayed, preparatory activities for the neural network and the application itself are running in the background.

When the app is launched for the first time, its assets are compiled to limit the amount of downloaded data. As a result, users may remain on this screen for a little longer while the app data is decompiled.

3.2 Main screen

On this screen, there are three buttons users can interact with. On the top left corner, there is a cogwheel icon, that takes users to settings. In the middle, there is a button "Choose from gallery", which is used, when the user already has the picture of the logo in their gallery, and the "Take the picture" button, which allows them to take a new picture.

3.3 File selector

The File Selector (also referred to as Filechooser) allows users to select an existing image for classification. While it is possible to use the file selector provided by the kivyMD library, I found it a bit confusing, so I went the route of calling the native file selector because it creates a familiar environment for the user and also makes them feel safer about the data they are providing to this application.

3.4 Camera screen

When a user wants to identify a logo they have found in real life, they usually don't have a picture of it in the gallery yet, so they end up on this screen. In the top left corner, you can see an arrow pointing to the left, which returns the user to the

main screen. In the middle of the screen is a camera preview with a pinch-to-zoom feature that allows users to see what they are about to photograph. And at the bottom center is a round camera button that, when pressed, saves the image to the user's gallery and starts image classification.

3.5 Result screen

Whether user selects a picture from the gallery or takes a new picture, they end up on the result screen. on the top left, there is a left-pointing arrow allowing them to get back to the main screen, and there are a few things in the main portion of the screen. From top to bottom, there is a visual representation of the classified logo, as well as the brand name just below. There is also an "incorrect logo" button in the bottom right corner, so if the shown logo doesn't match the actual logo, the user can send an input picture to my server, where it will later be used to retrain the neural network, and thus make the neural network more accurate in the future.

When the user reports incorrect classification, a so-called "snackbar" will be shown on the bottom of the screen for three seconds, informing the user about a successful report and thanking them for their contribution.

3.6 Settings screen

This screen is currently mainly for people like me who like a dark theme for battery savings and increased eye comfort. In addition to theme settings, there is a developer mode. This displays information such as the path to the image, resolution, class number, and probability on the screen. This information is not intended for normal users, but I left it in there as an option in case there is a bug on some devices, allowing me to get some debugging information from users.

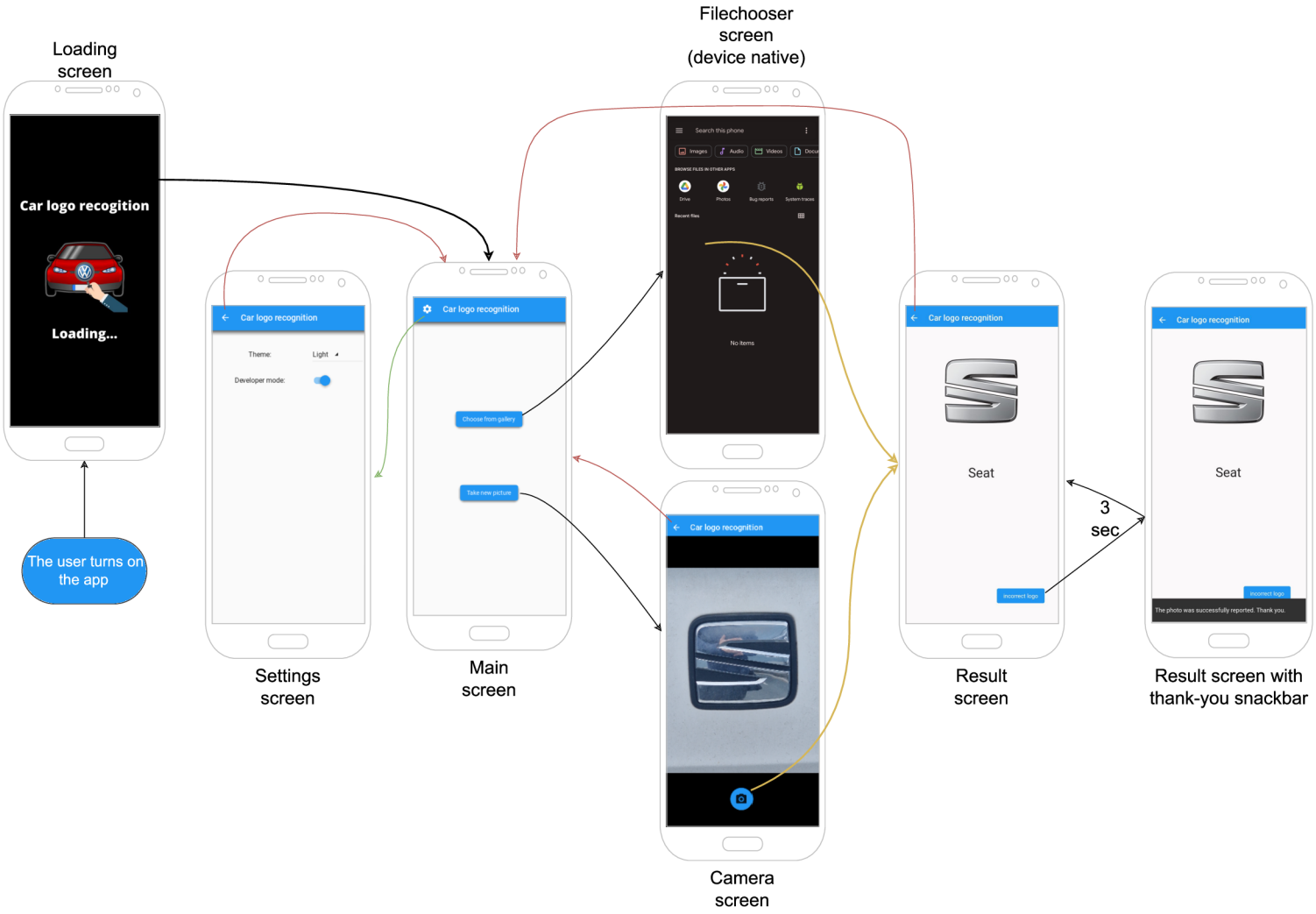

Fig. 3.1: Diagram of navigation through the app.

Conclusion

In this thesis, I first focused on finding the most suitable neural network for car logo recognition and gathering photos of car brands.

I took about 500 photos and downloaded 1800 photos from CompCars and 2900 photos from Kaggle datasets. This results in approximately 5200 photos for training my neural network. After training and testing the neural network, I found that for some brands, the dataset did not contain enough photos for reliable detection. This problem should be solvable by taking a higher number of photos and retraining the existing neural network.

I developed an Android app and published it in the Google Play app store. The app works mostly offline, however, certain features, such as reporting incorrectly recognized pictures, require an internet connection. The speed of recognition is not the greatest, but I'd consider it acceptable. It could be improved if the app would be optimized for each platform, but since the goal was multiplatform development, I settled with some drawbacks, like a larger app size or slower speed.

I planned to develop an iOS app as well, but I got stuck due to the compatibility issues, as explained in subsection 2.4.2. If this issue gets resolved at some point in the future, the complete layout and most of the code can be used as is, thanks to the use of the multiplatform Kivy framework. However, some platform-specific parts of the code, like permission handling, network checks, etc. would have to be written from scratch, because I wasn't able to successfully convert my code into an Xcode project, and therefore I couldn't access the necessary debugging information for me to write the code.

My initial approach of creating a convolutional neural network and converting it to tflite proved to not be feasible, for several reasons. First is the accuracy problem. I created a test dataset consisting of 20 photos for each of the 23 car brands (460 in total), selected from the VLD-45 dataset [17] that was not used for the training of neural network. Using all available photos, CNN did not reach the expected quality, but only about 65% on this test dataset. Second, converting the convolutional network to tflite proved to be highly problematic, due to limitations caused by the architecture and computational power of the device.

Since my original approach failed, I decided to create a new neural network directly in the tflite format. This neural network is based on the pre-trained Keras sequential_1 model, to which the dataset images are post-trained. This network achieved better results than the previously mentioned convolutional network, reaching approximately 80% accuracy on the same dataset. However, I think this result is due to the fact that this neural network is based on an already pre-trained model, and the success rate of the convolutional neural network could at least match this

result if more photos were used. The disadvantage of this approach is about 20 times longer classification time, compared to a tensorflow model.

One of the options I initially considered was also hosting a server running Python with the TensorFlow model loaded, and the mobile app would send the images to this server, which would process and classify them. It could then return the result back to the mobile app, which would display the recognized brand. I didn't take this route originally because I wanted the app to work offline, and I didn't want to worry about server scalability if more users started using my app. In retrospect, I can see that this might have been a better choice that would have saved me the trouble of trying to convert a trained AI model.

Bibliography

1. SICHA, Marek. *Detekce dopravních značek v reálném čase*. 2021. Available also from: https://www.vut.cz/studenti/zav-prace/detail/133597. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky. Vedoucí práce: Ing. Tomáš Bravenec.

2. PICTON, Phil. What is a Neural Network? In: *Introduction to Neural Networks* [online]. London: Macmillan Education UK, 1994 [visited on 2022-11-21]. ISBN 978-1-349-13530-1. Available from DOI: 10.1007/978-1-349-13530-1_1.

3. GUPTA, Dishashree. *25 must know concepts for beginners in Deep Learning &Neural Network* [online]. 2019. [visited on 2022-12-29]. Available from: https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/.

4. MA, Yaoyao; XU, Xiaoyu; YAN, Shuai; REN, Zhuoxiang. *A preliminary study on the resolution of electro-thermal multi-physics coupling problem using physics-informed Neural Network (PINN)* [online]. Multidisciplinary Digital Publishing Institute, 2022 [visited on 2022-12-15]. Available from: https://www.mdpi.com/1999-4893/15/2/53.

5. ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)* [online]. 2017, pp. 1–6 [visited on 2022-11-21]. Available from DOI: 10.1109/ICEngTechnol.2017.8308186.

6. ELGENDY, Mohammed. *3 convolutional neural networks (cnns) · Deep Learning for Vision Systems* [online]. 2020. [visited on 2022-12-15]. Available from: https://livebook.manning.com/book/deep-learning-for-vision-systems/chapter-3/v-7/117.

7. KOCH, Robert. *Machine learning datasets - definition, applications, resources* [online]. 2022. [visited on 2022-11-28]. Available from: https://www.clickworker.com/customer-blog/machine-learning-datasets/.

8. RIZVI, Mohd Sanad Zaki. *CNN image classification: Image Classification using CNN* [online]. 2020. [visited on 2022-12-15]. Available from: https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/.

9. SUSMAGA, Robert. Confusion Matrix Visualization. In: *Intelligent Information Processing and Web Mining*. Springer Berlin Heidelberg, 2004, pp. 107–116. Available from DOI: 10.1007/978-3-540-39985-8_12.

10. NARAYANAN, Venkatraman. *Proxemo: Gait-based emotion learning and Multi-view proxemic fusion for socially-aware Robot Navigation* [online]. 2020. [visited on 2022-12-28]. Available from: https://deepai.org/publication/proxemo-gait-based-emotion-learning-and-multi-view-proxemic-fusion-for-socially-aware-robot-navigation.

11. UCHYTIL, Tomas. *Car Logo Dataset* [Online]. 2023. [visited on 2023-05-27]. Available from: https://mega.nz/folder/wvw00TxQ#ZlYOlik9bRcbirvSJmAGNA.

12. YANG, Linjie; LUO, Ping; LOY, Chen Change; TANG, Xiaoou. A large-scale car dataset for fine-grained categorization and verification. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2015, pp. 3973–3981 [visited on 2022-10-11]. Available from DOI: 10.1109/CVPR.2015.7299023.

13. ÖZDEMIR, Volkan. *Car brand logos* [online]. 2021. [visited on 2022-10-10]. Available from: https://www.kaggle.com/datasets/volkandl/car-brand-logos?resource=download.

14. ÖZDEMIR, Volkan. *car logo classification* [online]. 2020. [visited on 2022-12-28]. Available from: https://www.kaggle.com/code/jp0909/car-logo-classification/notebook.

15. BAROT, Parth. *Comparing kivy and BeeWare: Understanding the key differences* [online]. 2023. [visited on 2023-04-01]. Available from: https://www.botreetechnologies.com/blog/kivy-vs-beeware-difference/.

16. MORASCHINELLI, Walter. *Picking a python mobile app development framework* [online]. 2023. [visited on 2023-04-01]. Available from: https://www.pangea.ai/dev-python-resources/picking-a-python-mobile-app-development-framework/.

17. YANG, Shuo; BO, Chunjuan; ZHANG, Junxing; GAO, Pengxiang; LI, Yujie; SERIKAWA, Seiichi. VLD-45: A Big Dataset for Vehicle Logo Recognition and Detection. *IEEE Transactions on Intelligent Transportation Systems* [online]. 2022, vol. 23, no. 12, pp. 25567–25573 [visited on 2023-05-25]. Available from DOI: 10.1109/TITS.2021.3062113.

Symbols and abbreviations

CNN Convolutional neural network

ANN Artificial neural network

AI Artificial intelligence

Tflite Tensorflow lite

A Content of the electronic attachment

- / ..... root directory of the attached archive
  - AI_model ..... latest tflite AI model
    - model_4_2.tflite
    - tensorflow_CNN_model.url
  - app ..... app source files and precompiled binary
    - noncompiled
      - camerax_provider ..... support files for app camera
        - ...
      - data ..... app icon, app splashscreen, labels and demo image of each brand
        - ...
      - buildozer.spec ..... configuration for Android compilation
      - layout.kv ..... app layout configuration
      - main.py ..... app source code
    - compiled.url ..... link to download compiled apk
    - Play_store_link.url
  - datasets ..... links to all datasets used in thesis
    - CompCars.url
    - Kaggle.url
    - my_photos.url
    - VLD-45.url
  - python_scripts ..... scripts used for thesis
    - convert_tensorflow_to_tflite.py
    - crop_all_by_third_from_each_side.py ... used for pre-crop to save some time
    - sort_CompCars_dataset.py ... for filtering and sorting CompCars dataset by brands
    - sort_VLD_dataset.py ..... for sorting VLD dataset by brands
    - tensorflow_classifier.py ..... would be used, if classification changes from in-app to server
    - tensorflow_classifier_train.py ..... for creating tensorflow model
    - tensorflow_confusion_matrix.py ..... for testing model performance
    - tflite_confusion_matrix.py ..... for testing tflite model performance
    - tflite_model_maker.py ..... for creating tflite model